# Intersection of Ellipses

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/


Created: October 10, 2000
Last Modified: June 23, 2015

## Contents

# 1 Introduction

This article describes how to compute the points of intersection of two ellipses, a geometric process referred to as a *find-intersection query*. It also shows how to determine whether two ellipses intersect without computing the points of intersection, a geometric process referred to as a *test-intersection query*. Specifically, the queries for the ellipses $E_0$ and $E_1$ are:

- *Find-intersection query*. The ellipses are considered to be hollow. If $E_0$ and $E_1$ intersect, find the points of intersection.
- *Test-intersection query*. The ellipses are considered to be solid. Determine whether
  - $E_0$ and $E_1$ are separated;
  - $E_0$ and $E_1$ overlap, where each ellipse contains points that the other ellipse does not;
  - $E_0$ is outside $E_1$, but they have a point in common on the elliptical boundaries;
  - $E_0$ strictly contains $E_1$ (no elliptical boundary points in common);
  - $E_0$ contains $E_1$, but they have a point in common on the elliptical boundaries;
  - $E_1$ strictly contains $E_0$ (no elliptical boundary points in common);
  - $E_1$ contains $E_0$, but they have a point in common on the elliptical boundaries; or
  - $E_0$ and $E_1$ are equal.

When two ellipses do not intersect, an implementation of the find-intersection query might not necessarily determine whether one ellipse is contained in the other or whether the two ellipses are separated. The find-intersection query treats the ellipses as hollow objects (just the curves). The test-intersection query treats the ellipses as solid objects.

In order to obtain robustness in an implementation, the find-intersection query uses quadratic equations to represent the ellipses. The coefficients are floating-point numbers that have exact rational representations. The algorithm uses rational arithmetic for various subproblems but computing with floating-point arithmetic only when necessary. In particular, the algorithm involves computing roots of low-degree polynomials. The classification of roots regarding domain (real or non-real) and multiplicity can be determined correctly using rational arithmetic. The closed-form expressions for the roots involve several functions that generally produce inexact results for rational inputs (square root, cube root, sine, cosine, arctangent). The mixed-type approach helps make the intersection construction robust as compared to using only floating-point operations throughout the algorithm.

# 2 Ellipse Representations

Given an ellipse in a standard form, we can convert it to a quadratic equation. The conversion in the opposite direction requires slightly more work. We must determine whether the equation has solutions and in fact represents an ellipse rather than a parabola or a hyperbola.

## 2.1 The Standard Form for an Ellipse

Let the ellipse center be $\mathbf{C}_0$. Let the ellipse axis directions be $\mathbf{U}_0$ and $\mathbf{U}_1$, a pair of unit-length orthogonal vectors. Let the ellipse extents along those axes be $\ell_0$ and $\ell_1$, a pair of positive numbers, each measuring the distance from the center to an extreme point along the corresponding axis. An ellipse point $\mathbf{P} = (x, y)$ is represented in the coordinate system $\{\mathbf{C}; \mathbf{U}_0, \mathbf{U}_1\}$ by $\mathbf{P} = \mathbf{C} + \xi_0\mathbf{U}_0 + \xi_1\mathbf{U}_1$, where $(\xi_0/\ell_0)^2 + (\xi_1/\ell_1)^2 = 1$. We can compute the coordinates by projection, $\xi_i = \mathbf{U}_i \cdot (\mathbf{P} - \mathbf{C})$, in which case

$$\begin{aligned} 1 &= (\xi_0/\ell_0)^2 + (\xi_1/\ell_1)^2 \\ &= (\mathbf{U}_0 \cdot (\mathbf{P} - \mathbf{C})/\ell_0)^2 + (\mathbf{U}_1 \cdot (\mathbf{P} - \mathbf{C})/\ell_1)^2 \\ &= (\mathbf{P} - \mathbf{C})^{\mathsf{T}} \left(\mathbf{U}_0\mathbf{U}_0^{\mathsf{T}}/\ell_0^2 + \mathbf{U}_0\mathbf{U}_1^{\mathsf{T}}/\ell_1^2\right) (\mathbf{P} - \mathbf{C}) \\ &= (\mathbf{P} - \mathbf{C})^{\mathsf{T}} M (\mathbf{P} - \mathbf{C}) \end{aligned} \tag{1}$$

where the last equality defines the positive definite matrix $M$. By definition, such a matrix is symmetric, has positive eigenvalues, and is invertible.

## 2.2 Conversion to a Quadratic Equation

We can expand the standard equation (1) into a quadratic equation

$$k_0 + k_1 x + k_2 y + k_3 x^2 + k_4 xy + k_5 y^2 = 0 \tag{2}$$

where the $k_i$ depend on components of $\mathbf{C}$, $\mathbf{U}_0$, $\mathbf{U}_1$, $\ell_0$, and $\ell_1$. Conversely, if we are given the coefficients of a quadratic equation (2), we can factor to the standard equation (1). The quadratic equation in vector-matrix form is

$$\mathbf{P}^{\mathsf{T}} A \mathbf{P} + \mathbf{B}^{\mathsf{T}} \mathbf{P} + k_0 = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} k_3 & k_4/2 \\ k_4/2 & k_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} k_1 & k_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + k_0 = 0 \tag{3}$$

where $A$ and $\mathbf{B}$ are defined by the first equality. To complete the square, observe that

$$(\mathbf{P} - \mathbf{C})^{\mathsf{T}} A (\mathbf{P} - \mathbf{C}) = \mathbf{P}^{\mathsf{T}} A \mathbf{P} - 2\mathbf{C}^{\mathsf{T}} A \mathbf{P} + \mathbf{C}^{\mathsf{T}} A \mathbf{C} \tag{4}$$

To match the linear term of equation (3), we need $-2A\mathbf{C} = \mathbf{B}$ and $\mathbf{C}^{\mathsf{T}} A \mathbf{C} = k_0$. To represent an ellipse, minimally $A$ must be invertible (among other conditions), so $\mathbf{C} = -A^{-1}\mathbf{B}/2$ which leads to

$$(\mathbf{P} - \mathbf{C})^{\mathsf{T}} A (\mathbf{P} - \mathbf{C}) = \mathbf{B}^{\mathsf{T}} A^{-1} \mathbf{B}/4 - k_0 \tag{5}$$

The right-hand side must be nonzero to obtain the standard form of equation (1). If so, then divide through by that expression to obtain

$$(\mathbf{P} - \mathbf{C})^{\mathsf{T}} M (\mathbf{P} - \mathbf{C}) = (\mathbf{P} - \mathbf{C})^{\mathsf{T}} \frac{A}{\mathbf{B}^{\mathsf{T}} A^{-1} \mathbf{B}/4 - k_0} (\mathbf{P} - \mathbf{C}) = 1 \tag{6}$$

where the first equality defines the matrix $M = [m_{ij}]$. To represent an ellipse, $M$ must be positive definite, which is equivalent to $m_{00} > 0$, $m_{11} > 0$, and $m_{00}m_{11} - m_{01}^2 > 0$. The matrix can then be factored as

$$M = d_0 \mathbf{U}_0 \mathbf{U}_0^{\mathsf{T}} + d_1 \mathbf{U}_0 \mathbf{U}_1^{\mathsf{T}} = R D R^{\mathsf{T}} \tag{7}$$

where $R = [\mathbf{U}_0\ \mathbf{U}_1]$ is an orthogonal matrix whose columns are the specified unit-length vectors and where $D = \text{Diag}(d_0, d_1)$ is a diagonal matrix with positive diagonal entries. The ellipse axes are the eigenvectors of $M$ and the ellipse axis extents are $\ell_i = 1/\sqrt{d_i}$.

Unfortunately, factoring $M$ using an eigensolver cannot be performed with only rational arithmetic. A goal of this document is to try to avoid floating-point rounding errors until the very end of the process when the root finders must call sqrt, pow, sin, or cos. As it turns out, it is possible to determine whether the quadratic equation represents an ellipse using only rational arithmetic. The condition $m_{00}m_{11} - m_{01}^2 > 0$ implies $k_4^2/4 < k_3k_5$, and consequently $k_3 \neq 0$ and $k_5 \neq 0$. We may divide the quadratic equation by $k_5$; for simplicity, let us just think of it as 1 and rename the divided coefficients,

$$0 = k_0 + k_1x + k_2y + k_3x^2 + k_4xy + y^2 = (k_0 + k_1x + k_3x^2) + (k_2 + k_4x)y + y^2 \tag{8}$$

with $k_3 - k_4^2/4 > 0$. The $y$-term may be eliminated by the transformation $y = w - (k_2 + k_4x)/2$,

$$0 = w^2 + (k_0 - k_2^2/4) + (k_1 - k_2k_4/2)x + (k_3 - k_4^2/4)x^2 \tag{9}$$

The last three terms form a quadratic polynomial in $x$ that must be negative for some interval of $x$-values in order that the sum with $w^2$ be zero. Therefore, the polynomial must have two distinct real-valued roots $x_0 < x_1$ and the ellipse points are generated for $x \in [x_0, x_1]$. To have distinct real roots, it is required that $(k_1 - k_2k_4/2)^2 - 4(k_0 - k_2^2/4)(k_3 - k_4^2/4) > 0$.

### 2.3 Conversion to a Quadratic Equation Without Normalization

The previous section started with a standard-form ellipse, $(\mathbf{P}-\mathbf{C})^\mathsf{T}M(\mathbf{P}-\mathbf{C}) = 1$, where $M = \mathbf{U}_0\mathbf{U}_0^\mathsf{T}/\ell_0^2 + \mathbf{U}_1\mathbf{U}_1^\mathsf{T}/\ell_1^2$. The unit-length vectors $\mathbf{U}_0$ and $\mathbf{U}_1$ are the ellipse axis directions. In practice, the axis directions might be obtained from vectors $\mathbf{V}_i$ by normalizing, $\mathbf{U}_i = \mathbf{V}_i/|\mathbf{V}_i|$. The normalization generally cannot be performed without rounding errors because of the implied square root when computing the length of $\mathbf{V}_i$. To avoid introducing this error into the inputs of the intersection query, use the ellipse representation

$$(\mathbf{P}-\mathbf{C})^\mathsf{T}\left(\frac{\mathbf{V}_0\mathbf{V}_0^\mathsf{T}}{\ell_0^2|\mathbf{V}_0|^2} + \frac{\mathbf{V}_1\mathbf{V}_1^\mathsf{T}}{\ell_1^2|\mathbf{V}_1|^2}\right)(\mathbf{P}-\mathbf{C}) = 1 \tag{10}$$

The conversion from this representation to a quadratic equation of the form of equation (2) involves only rational operations, so no rounding errors are introduced because of normalization.

## 3 Find-Intersection Query

### 3.1 Constructing Intersection Points

The ellipses may be written as quadratic equations, where we assume that $a_5 = b_5 = 1$. In an implementation, the caller can pass $a_5$ and $b_5$ and the division by these numbers is performed internally. Let the equations be

$$\begin{aligned} A(x,y) &= a_0 + a_1x + a_2y + a_3x^2 + a_4xy + y^2 = 0 \\ B(x,y) &= b_0 + b_1x + b_2y + b_3x^2 + b_4xy + y^2 = 0 \end{aligned} \tag{11}$$

According to the last section, these define ellipses when $a_3 - a_4^2/4 > 0$, $b_3 - b_4^2/4 > 0$, $(a_1 - a_2a_4/2)^2/4 - (a_0 - a_2^2/4)(a_3 - a_4^2/4) > 0$, and $(b_1 - b_2b_4/2)^2/4 - (b_0 - b_2^2/4)(b_3 - b_4^2/4) > 0$.

Subtract the two equations to obtain the quadratic equation

$$D(x, y) = d_0 + d_1x + d_2y + d_3x^2 + d_4xy = 0 \tag{12}$$

where $d_i = a_i - b_i$ for all $i$. If $d_i = 0$ for all $i$, then the two equations are the same and an implementation should report that the ellipses are identical.

Assuming not all $d_i = 0$, define the transformation $y = w - (a_2 + a_4x)/2$, which eliminates the $y$-term in $A(x, y) = 0$,

$$0 = w^2 + (a_0 + a_1x + a_3x^2) - (a_2 + a_4x)^2/4 = w^2 + (c_0 + c_1x + c_2x^2) = w^2 + c(x) \tag{13}$$

where the last two equalities define $c(x) = c_0 + c_1x + c_2x^2$ with $c_0 = a_0 - a_2^2/4$, $c_1 = a_1 - a_2a_4/2$, and $c_2 = a_3 - a_4^2/4 > 0$. Substituting the change of variables into $D(x, y) = 0$, we obtain

$$0 = (d_2 + d_4x)w + (e_0 + e_1x + e_2x^2) = d(x)w + e(x) \tag{14}$$

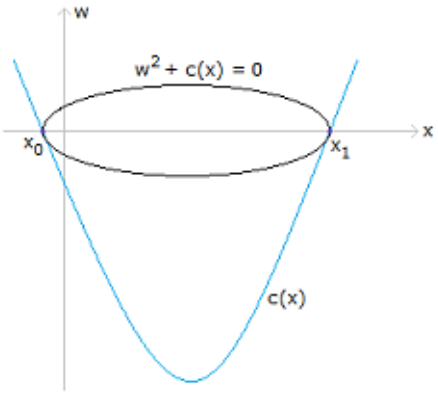
where the last equality defines $d(x) = d_2 + d_4x$ and $e(x) = e_0 + e_1x + e_2x^2$ with $e_0 = d_0 - a_2d_2/2$, $e_1 = d_1 - (a_2d_4 + a_4d_2)/2$, and $e_2 = d_3 - a_4d_4/2$. The intersection points are pairs $(x, w)$ that solve equations (13) and (14) simultaneously. The algorithm for solving the equations has several cases that require branching on coefficients $d_i$; it is helpful to motivate the cases geometrically.

Equation (13) can be factored to standard form,

$$\frac{\left(x + \frac{c_1}{2c_2}\right)^2}{\frac{c_1^2 - 4c_0c_2}{4c_2^2}} + \frac{w^2}{\frac{c_1^2 - 4c_0c_2}{4c_2}} = 1 \tag{15}$$

As mentioned previously, in order for this equation to have solutions it is necessary that $c_2 > 0$ and $c_1^2 - 4c_0c_2 > 0$. The $x$-values for which the ellipse is defined is the interval $[x_0, x_1]$, where $x_0$ and $x_1$ are the roots of $c(x)$; that is, $c(x) \le 0$ for $x \in [x_0, x_1]$ in which case $w^2 = -c(x)$ has real-valued solutions for $w$. Figure 1 shows the graph of a typical ellipse and the corresponding graph for $c(x)$.

**Figure 1.** The graph of the ellipse $w^2 + c(x) = 0$ (in black) and the graph of $c(x)$ (in blue). The roots of $c(x)$ are $x_0$ and $x_1$. The ellipse is defined for $x \in [x_0, x_1]$.



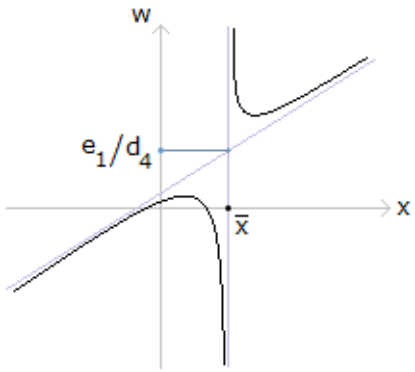
The analysis of equation (14) includes several cases that involve the coefficient $d(x) = d_2 + d_4x$ of $w$ and whether $e(x)$ is a linear or a quadratic polynomial. In the cases, define $\bar{x} = -d_2/d_4$. I also discuss whether an intersection is *transverse*, where the graph tangents are not parallel (the graphs cut through each other). If not transverse, the intersection is *tangential*, where the graph tangents are parallel. Equivalently, the graph normals are parallel at the intersection, and we can use gradients of functions for the normals when the curves are defined implicitly.

#### 3.1.1 Case $d_4 \neq 0$ and $e(\bar{x}) \neq 0$

When $d_4 \neq 0$ and $e_2 \neq 0$, equation (14) represents a hyperbola with a vertical asymptote $x = \bar{x}$. The other asymptote is $w = (e_2/d_4)x + (e_1d_4 - d_2e_2)/d_4^2$. Figure 2 shows a typical graph.

---

**Figure 2.** A typical graph of the hyperbola defined by equation (14) when $d_4 \neq 0$ and $e(\bar{x}) \neq 0$. The nonhorizontal asymptote occurs when $e_2 \neq 0$. If $e_2 = 0$, $e(x)$ is linear and the asymptote is horizontal.




---

In fact, the graph is that of the function $w = -e(x)/d(x)$. Substituting this into equation (13), we obtain the quartic polynomial $f(x) = c(x)d^2(x) + e^2(x)$ for which $\bar{x}$ is not a root.

The intersections of the hyperbola and the axis-aligned ellipse correspond to intersections for the original ellipses. It is possible to determine whether the intersection is transverse or tangential. The gradient of the function $w^2 + c(x)$ is $(c'(x), 2w)$ and the gradient of the function $d(x)w + e(x)$ is $(d'(x)w + e'(x), d(x))$. To be parallel, the dot-perp of the gradients must be zero,

$$\begin{aligned}
0 &= (c'(x), 2w) \cdot (d'(x)w + e'(x), d(x))^{\perp} \\
&= (c'(x), 2w) \cdot (d(x), -d'(x)w - e'(x)) \\
&= c'(x)d(x) - 2d'(x)w^2 - 2e'(x)w \\
&= c'(x)d(x) + 2d'(x)c(x) + 2e(x)e'(x)/d(x) \qquad \text{[using equations (13) and (14)]} \\
&= (c'(x)d^2(x) + 2c(x)d(x)d'(x) + 2e(x)e'(x))/d(x) \\
&= f'(x)/d(x)
\end{aligned} \tag{16}$$

If the intersection is tangential for some $\hat{x}$, then $f(\hat{x}) = f'(\hat{x}) = 0$; that is, it is necessary that $\hat{x}$ be a repeated root for $f(x)$. In fact, this is also a sufficient condition based on the geometry of the graphs.

Pseudocode for this case is provided by Listing 1.

---

**Listing 1.** An implementation for the case $d_4 \neq 0$ and $e(\bar{x}) \neq 0$. The graph of $w = -e(x)/d(x)$ is a hyperbola and the points of intersection with the ellipse are generated by roots to the quartic polynomial $f(x)$.

```
int numPoints = 0;
Vector2<Real> intersection[4];
bool isTransverse[4];

Real f0 = c0 * d2 * d2 + e0 * e0;
Real f1 = c1 * d2 * d2 + 2 * (c0 * d2 * d4 + e0 * e1);
Real f2 = c2 * d2 * d2 + c0 * d4 * d4 + e1 * e1 + 2 * (c1 * d2 * d4 + e0 * e2);
Real f3 = c1 * d4 * d4 + 2 * (c2 * d2 * d4 + e1 * e2);
Real f4 = c2 * d4 * d4 + e2 * e2;  // > 0

map<Real, int> rmMap;  // (root, multiplicity)
SolveQuartic(f0, f1, f2, f3, f4, rmMap);
for_each (rm in rmMap)
{
    Real x = rm.first;
    int multiplicity = rm.second;
    Real w = -(e0 + x * (e1 + x * e2)) / (d2 + d4 * x);
    Real y = w - (a2 + x * a4) / 2;
    intersection[numPoints] = { x, y };
    isTransverse[numPoints] = (multiplicity == 1);
    ++numPoints;
}
```

---

The function SolveQuartic must compute the theoretically correct multiplicity of the root. The GTEngine quartic root finder satisfies this constraint when Real is a rational arithmetic type.

### 3.1.2 Case $d_4 \neq 0$ and $e(\bar{x}) = 0$

When $e(\bar{x}) = 0$, the hyperbola of the previous section degenerates to a pair of lines. In a sense, the asymptotes of the hyperbola are the limits of the hyperbolic curves as the polynomial coefficients of $e(x)$ vary from a state where $e(\bar{x}) \neq 0$ to a state where $e(\bar{x}) = 0$. The polynomial factors to $e(x) = (d_2 + d_4 x)(h_0 + h_1 x)$, where $h_1 = e_2/d_4$ and $h_0 = (e_1 - d_2 h_1)/d_4$. The lines are $x = \bar{x}$ and $w = -(h_0 + h_1 x)$. Define $h(x) = h_0 + h_1 x$.

An intersection of $x = \bar{x}$ with the ellipse $w^2 + c(x) = 0$ occurs when $c(\bar{x}) \leq 0$, in which case $w = \pm\sqrt{-c(\bar{x})}$. The intersection is transverse when $c(\bar{x}) < 0$ or tangential when $c(\bar{x}) = 0$. These sign tests can be computed accurately when using rational arithmetic.

An intersection of the line $w + h(x) = 0$ with the ellipse $w^2 + c(x) = 0$ occurs at roots of the quadratic polynomial $f(x) = h^2(x) + c(x)$. The intersection is tangential when the gradients at the intersection are parallel. The gradient of $w^2 + c(x)$ is $(c'(x), 2w)$ and the gradient of $w + h(x)$ is $(h'(x), 1)$. To be parallel,

the dot-perp of the gradients must be zero,

$$\begin{aligned} 0 &= (c'(x), 2w) \cdot (h'(x), 1)^{\perp} \\ &= (c'(x), 2w) \cdot (1, -h'(x)) \\ &= c'(x) - 2h'(x)w \\ &= c'(x) + 2h(x)h'(x) \\ &= f'(x) \end{aligned} \tag{17}$$

The intersection is tangential for some $\hat{x}$ iff $f(\hat{x}) = f'(\hat{x}) = 0$; that is, it is necessary that $\hat{x}$ be a repeated root for $f(x)$.

Pseudocode for this case is provided by Listing 2.

---

**Listing 2.** An implementation for the case $d_4 \neq 0$ and $e(\bar{x}) = 0$. The graph consists of two lines and the points of intersection with the ellipse are generated by roots to the quadratic polynomial $f(x)$.

```
int numPoints = 0;
Vector2<Real> intersection[4];
bool isTransverse[4];
Real translate, w, y;

// Compute intersections of x = xbar with ellipse.
Real xbar = -d2 / d4, ncbar = -(c0 + xbar * (c1 + xbar * c2));
if (ncbar >= 0)
{
    translate = (a2 + xbar * a4) / 2;
    w = sqrt(ncbar);
    y = w - translate;
    intersection[numPoints] = { xbar, y };
    if (w > 0)
    {
        isTransverse[numPoints++] = true;
        w = -w;
        y = w - translate;
        intersection[numPoints] = { xbar, y };
        isTransverse[numPoints++] = true;
    }
    else
    {
        isTransverse[numPoints++] = false;
    }
}

// Compute intersections of w = -h(x) with ellipse.
Real h1 = e2 / d4;
Real h0 = (e1 - d2 * h1) / d4;
Real f0 = c0 + h0 * h0;
Real f1 = c1 + 2 * h0 * h1;
Real f2 = c2 + h1 * h1;  // > 0

map<Real, int> rmMap;  // (root, multiplicity)
SolveQuadratic(f0, f1, f2, rmMap);
for_each (rm kin rmMap)
{
    Real x = rm.first;
    int multiplicity = rm.second;
    translate = (a2 + xbar * a4) / 2;
    w = -(h0 + x * h1);
    y = w - translate;
    intersection[numPoints] = { x, y };
```

```
        isTransverse[numPoints++] = (multiplicity == 1);
    }
```

The function SolveQuadratic must compute the theoretically correct multiplicity of the root. The GTEngine quadratic root finder satisfies this constraint when Real is a rational arithmetic type.

### 3.1.3 Case $d_4 = 0$, $d_2 \neq 0$, and $e_2 \neq 0$

We may solve for $w$ directly in equation (14), $w = -e(x)/d_2$, where $e(x)$ is a quadratic polynomial ($e_2 \neq 0$). The graph of this equation is a parabola, and we want to compute intersections of the parabola with the ellipse. Substitute into equation (13) and multiply by $d_2^2$ to obtain the quartic polynomial $f(x) = d_2^2 c(x) + e^2(x)$. The roots of $f(x)$ lead to intersections of the parabola and ellipse.

An intersection is tangential when the gradients at the intersection are parallel. The gradient of $w^2 + c(x)$ is $(c'(x), 2w)$ and the gradient of $d_2 w + e(x)$ is $(e'(x), d_2)$. To be parallel, the dot-perp of the gradients must be zero,

$$\begin{aligned} 0 &= (c'(x), 2w) \cdot (e'(x), d_2)^\perp \\ &= (c'(x), 2w) \cdot (d_2, -e'(x)) \\ &= d_2 c'(x) - 2e'(x)w \\ &= d_2 c'(x) + 2e(x)e'(x)/d_2 \\ &= f'(x)/d_2^2 \end{aligned} \tag{18}$$

The intersection is tangential for some $\hat{x}$ iff $f(\hat{x}) = f'(\hat{x}) = 0$; that is, it is necessary that $\hat{x}$ be a repeated root for $f(x)$.

Pseudocode for this case is provided by Listing 3.

**Listing 3.** An implementation for the case $d_4 = 0$, $d_2 \neq 0$, and $e_2 \neq 0$. The graph of $w = -e(x)/d_2$ is a parabola and the points of intersection with the ellipse are generated by roots to the quartic polynomial $f(x)$.

```
int numPoints = 0;
Vector2<Real> intersection[4];
bool isTransverse[4];

Real f0 = c0 * d2 * d2 + e0 * e0;
Real f1 = c1 * d2 * d2 + 2 * e0 * e1;
Real f2 = c2 * d2 * d2 + e1 * e1 + 2 * e0 * e2;
Real f3 = 2 * e1 * e2;
Real f4 = e2 * e2;  // > 0

map<Real, int> rmMap;  // (root, multiplicity)
SolveQuartic(f0, f1, f2, f3, f4, rmMap);
for_each (rm in rmMap)
{
    Real x = rm.first;
    int multiplicity = rm.second;
    Real w = -(e0 + x * (e1 + x * e2)) / d2;
```

```
        Real y = w - (a2 + x * a4) / 2;
        intersection[numPoints] = { x, y };
        isTransverse[numPoints] = (multiplicity == 1);
        ++numPoints;
    }
```

---

The function SolveQuartic must compute the theoretically correct multiplicity of the root. The GTEngine quartic root finder satisfies this constraint when Real is a rational arithmetic type.

### 3.1.4 Case $d_4 = 0$, $d_2 \neq 0$, and $e_2 = 0$

The analysis in the previous section applies equally well here, except that $e(x)$ now is a linear polynomial and the graph of $w = -e(x)/d_2$ is a line. The function whose roots lead to line-ellipse intersections is $f(x) = d_2 c(x) + e^2(x)$, which is a quadratic polynomial. Tangential intersections occur at repeated roots of $f(x)$.

Pseudocode for this case is provided by Listing 4.

---

**Listing 4.** An implementation for the case $d_4 = 0$, $d_2 \neq 0$, and $e_2 = 0$. The graph of $w = -e(x)/d_2$ is a line and the points of intersection with the ellipse are generated by roots to the quadratic polynomial $f(x)$.

```
int numPoints = 0;
Vector2<Real> intersection[4];
bool isTransverse[4];

Real f0 = c0 * d2 * d2 + e0 * e0;
Real f1 = c1 * d2 * d2 + 2 * e0 * e1;
Real f2 = c2 * d2 * d2 + e1 * e1;  // > 0

map<Real, int> rmMap;  // (root, multiplicity)
SolveQuadratic(f0, f1, f2, rmMap);
for_each (rm in rmMap)
{
    Real x = rm.first;
    int multiplicity = rm.second;
    Real w = -(e0 + x * e1) / d2;
    Real y = w - (a2 + x * a4) / 2;
    intersection[numPoints] = { x, y };
    isTransverse[numPoints] = (multiplicity == 1);
    ++numPoints;
}
```

---

The function SolveQuadratic must compute the theoretically correct multiplicity of the root. The GTEngine quadratic root finder satisfies this constraint when Real is a rational arithmetic type.

### 3.1.5 Case $d_4 = 0$ and $d_2 = 0$

Equation (14) is $e(x) = 0$, where $e(x)$ can be either quadratic or linear, and equation (13) is $w^2 + c(x) = 0$. For each root $\hat{x}$ of $e(x)$, if $c(\hat{x}) \leq 0$, we have corresponding solutions $\hat{w} = \pm\sqrt{-c(\hat{x})}$. The technical challenge

is to determine the sign of $c(\hat{x})$ accurately without misclassifications due to floating-point arithmetic.

The subcase when $e(x)$ is linear ($e_2 = 0$) is easy to solve. Suppose that $e_1 \neq 0$; then the only root is $\hat{x} = -e_0/e_1$. If $c(\hat{x}) < 0$, there are two transverse intersection points with $\hat{w} = \pm\sqrt{c(\hat{x})}$. If $c(\hat{x}) = 0$, there is one tangential intersection point with $\hat{w} = 0$. For the transverse intersections, when rational arithmetic is used we compute $\hat{x}$ without numerical errors. The computation $c(\hat{x})$ also has no numerical errors, so we know the sign of $c(\hat{x})$ exactly. This allows us to determine that the intersections are transverse. However, numerical error is (usually) introduced when computing the square root function. Thus, $\hat{x}$ is exact but $\hat{w}$ is approximate, even though we know the intersection is transverse. Pseudocode for this case is provided by Listing 5.

---

**Listing 5.** An implementation for the case $d_4 = 0$, $d_2 = 0$, and $e_2 = 0$.

```
int numPoints = 0;
Vector2<Real> intersection[4];
bool isTransverse[4];
Real w, y;

Real xhat = -e0 / e1;
Real nchat = -(c0 + xhat * (c1 + xhat * c2));
if (nchat > 0)
{
    Real translate = (a2 + xhat * a4) / 2;
    w = sqrt(nchat);
    y = w - translate;
    intersection[numPoints] = { xhat, y };
    isTransverse[numPoints++] = true;
    w = -w;
    y = w - translate;
    intersection[numPoints] = { xhat, y };
    isTransverse[numPoints++] = true;
}
else if (nchat == 0)
{
    y = -(a2 + xhat * a4) / 2;  // w = 0
    intersection[numPoints] = { xhat, y };
    isTransverse[numPoints++] = false;
}
```

---

When $e_1 = 0$, $e(x) = e_0$ is a constant polynomial. It cannot be zero, because if it were, all $d_i = 0$. However, an implementation can determine this condition early and exit immediately, reporting that the ellipses are the same.

The subcase when $e(x)$ is quadratic ($e_2 \neq 0$) is not as easy to solve. The main problem is computing the correct sign of $c(\hat{x})$ when $\hat{x}$ is a non-rational root of $e(x)$. Let's first make $e(x)$ monic by dividing through by $e_2$; define $f(x) = f_0 + f_1 x + x^2$, where $f_0 = e_0/e_2$ and $f_1 = e_1/e_2$. The roots are $\hat{x} = -f_1/2 \pm \sqrt{\Delta}$, where $\Delta = f_1^2/4 - f_0$. If $\Delta < 0$, there is nothing to do because there are no real-valued roots. If $\Delta = 0$, the root is repeated and rational, so we can compute the exact sign of $c(\hat{x})$.

If $\Delta > 0$, we have two distinct roots. Generally, we cannot expect $\Delta$ to be a squared rational, so the analysis must be conservative and assume that $\sqrt{\Delta}$ will involve numerical rounding errors. A sequence of algebraic manipulations involving inequalities, however, can tell us the correct sign of $c(\hat{x})$.

The condition to test is $c(\hat{x}) = c_0 + c_1\hat{x} + c_2\hat{x}^2 \leq 0$. Regardless of which root we want to test, we may

substitute $\hat{x}^2 = -(f_0 + f_1\hat{x})$ into the inequality to obtain $g_0 + g_1\hat{x} \le 0$, where $g_0 = c_0 - c_2 f_0$ and $g_1 = c_1 - c_2 f_1$. Finally, we must analyze $g_1\hat{x} \le -g_0$, which depends on the sign of $g_1$.

If $g_1 = 0$, the parabolas defined by $c(x)$ and $e(x)$ are the same shape; one is a vertical translation of the other. If $g_0 < 0$, the graph of $e(x)$ is above that of $c(x)$, so both roots $\hat{x}$ of $e(x)$ are in the interval whose endpoints are the roots of $c(x)$; that is, $c(\hat{x}) < 0$ for both roots. Moreover, the generated intersections are transverse. If $g_0 > 0$, the graph of $e(x)$ is below that of $e(x)$, so $c(\hat{x}) > 0$ for both roots and there are no real-valued solutions to $w^2 + c(x) = 0$, which implies there are no ellipse-ellipse intersections. If $g_0 = 0$, the graphs of $e(x)$ and $c(x)$ are the same, so $c(\hat{x}) = 0$ for both roots. The corresponding intersections are both tangential.

Suppose that $g_1 > 0$; we now want to analyze $\hat{x} \le -g_0/g_1$ for a root $\hat{x}$ of $e(x)$. For the root $\hat{x} = -f_1/2 + \sqrt{\Delta}$, we need $\sqrt{\Delta} \le -g_0/g_1 + f_1/2 = r$, where the last equality defines $r$. The statement is true when $r \ge 0$ and $\Delta \le r^2$. These conditions may all be computed using rational arithmetic without generating rounding errors. For the root $\hat{x} = -f_1/2 - \sqrt{\Delta}$, we need $\sqrt{\Delta} \ge -r$. The statement is true when $r > 0$ or when $r \le 0$ and $\Delta \ge r^2$. The choice of strict inequality versus nonstrict inequality is made to support the determination of transverse or tangential intersection.

Suppose that $g_1 < 0$; we now want to analyze $\hat{x} \ge -g_0/g_1$. For the root $\hat{x} = -f_1/2 - \sqrt{\Delta}$, we need $\sqrt{\Delta} \le -r$. The statement is true when $r \le 0$ and $\Delta \le r^2$. For the root $\hat{x} = -f_1/2 + \sqrt{\Delta}$, we need $\sqrt{\Delta} \ge r$. The statement is true when $r < 0$ or when $r \ge 0$ and $\Delta \ge r^2$.

Pseudocode for this case is provided by Listing 6. The function SpecialIntersection is used to compute $y$ from $\hat{x}$ and to set the condition about transversality.

---

**Listing 6.** An implementation for the case $d_4 = 0$, $d_2 = 0$, and $e_2 \neq 0$.

```
struct Result
{
    int numPoints;
    Vector2<Real> intersection[4];
    bool isTransverse[4];
};

Result result;
result.numPoints = 0;

Real f0 = e0 / e2, f1 = e1 / e2;
Real mid = -f1 / 2;
Real discr = mid * mid - f0;
if (discr > 0)
{
    // The roots are xhat = mid + s*sqrtDiscr for s in {-1,1}.
    Real sqrtDiscr = sqrt(discr);
    Real g0 = c0 - c2 * f0, g1 = c1 - c2 * f1;
    if (g1 > 0)
    {
        // We need s*sqrt(discr) <= -g0/g1 + f1/2.
        Real r = -g0 / g1 - mid;

        // s = +1:
        if (r >= 0)
        {
            Real rsqr = r * r;
            if (discr < rsqr)
            {
                SpecialIntersection(mid + sqrtDiscr, true, result);
            }
            else if (discr == rsqr)
```

```
            {
                SpecialIntersection(mid + sqrtDiscr, false, result);
            }
        }

        // s = -1:
        if (r > 0)
        {
            SpecialIntersection(mid - sqrtDiscr, true, result);
        }
        else
        {
            Real rsqr = r * r;
            if (discr > rsqr)
            {
                SpecialIntersection(mid - sqrtDiscr, true, result);
            }
            else if (discr == rsqr)
            {
                SpecialIntersection(mid - sqrtDiscr, false, result);
            }
        }
    }
    else if (g1 < 0)
    {
        // We need s*sqrt(discr) >= -g0/g1 + f1/2.
        Real r = -g0 / g1 - mid;

        // s = -1:
        if (r <= 0)
        {
            Real rsqr = r * r;
            if (discr < rsqr)
            {
                SpecialIntersection(mid - sqrtDiscr, true, result);
            }
            else
            {
                SpecialIntersection(mid - sqrtDiscr, false, result);
            }
        }

        // s = +1:
        if (r < 0)
        {
            SpecialIntersection(mid + sqrtDiscr, true, result);
        }
        else
        {
            Real rsqr = r * r;
            if (discr > rsqr)
            {
                SpecialIntersection(mid + sqrtDiscr, true, result);
            }
            else if (discr == rsqr)
            {
                SpecialIntersection(mid + sqrtDiscr, false, result);
            }
        }
    }
    else  // g1 = 0
    {
        // The graphs of c(x) and f(x) are parabolas of the same
        // shape.  One is a vertical translation of the other.
        if (g0 < 0)
        {
            // The graph of f(x) is above that of c(x).
            SpecialIntersection(mid - sqrtDiscr, true, result);
            SpecialIntersection(mid + sqrtDiscr, true, result);
        }
        else if (g0 == 0)
        {
```

```
            // The graphs of c(x) and f(x) are the same parabola.
            SpecialIntersection(mid - sqrtDiscr, false, result);
            SpecialIntersection(mid + sqrtDiscr, false, result);
        }
        // else: graph of f(x) is below that of c(x), no intersections
    }
}
else if (discr == 0)
{
    // The theoretical root of e(x) is x = -e1/2.
    Real nchat = -(c0 + mid * (c1 + mid * c2));
    if (nchat > 0)
    {
        SpecialIntersection(mid, true, result);
    }
    else if (nchat == 0)
    {
        SpecialIntersection(mid, false, result);
    }
}

void SpecialIntersection(Real x, bool transverse, Result& result)
{
    if (transverse)
    {
        Real translate = (a2 + x * a4) / 2;
        Real nc = -(c0 + x * (c1 + x * c2));
        if (nc < 0)
        {
            // Clamp to eliminate the rounding error, but duplicate the point
            // because we know that it is a transverse intersection.
            nc = 0;
        }

        Real w = sqrt(nc);
        Real y = w - translate;
        result.intersection[result.numPoints] = { x, y };
        result.isTransverse[result.numPoints++] = true;
        w = -w;
        y = w - translate;
        result.intersection[result.numPoints] = { x, y };
        result.isTransverse[result.numPoints++] = true;
    }
    else
    {
        // The vertical line at the root is tangent to the ellipse.
        Real y = -(a2 + x * a4) / 2;  // w = 0
        result.intersection[result.numPoints] = { x, y };
        result.isTransverse[result.numPoints++] = false;
    }
}
```

---

## 3.2 An Implementation

In GTEngine, the polynomial root finders are implemented in file GteRootsPolynomial.h. The root functions SolveQuartic and SolveQuadratic must compute the roots with the correct classification (real or non-real) and multiplicity. If the root finders were to return the same root multiple times, extra logic would have to be added after the calls to test for equal roots and discard duplicates; otherwise, the array of intersection points might overflow.

The find-intersection query is implemented in the file GteIntrEllipse2Ellipse2.h. In this code, the inputs are

ellipses, so the tests for invalid input coefficient arrays are skipped. The code converts the ellipses to coefficient arrays internally.

# 4 Test-Intersection Query

At first glance, one might consider using the algorithm for the find-intersection query but without root finding; that is, set up the polynomials $f(x)$ and count the number of real-valued roots, something that is presumably faster than actually computing the roots. The problem is that there can be cases where $\hat{x}$ is a root of $f(x)$, but there is no corresponding $\hat{y}$. For example, we can have a root $\hat{x}$ and need to solve $w^2 = c(\hat{x}) < 0$, but there is no real-valued solution $\hat{w}$ and, therefore, no real-valued solution $\hat{y} = \hat{w} - (a_2 + a_4\hat{x})/2$. Root counting appears to be insufficient for a test-intersection query.

A different approach is discussed in this section. The idea is to transform the ellipses, using an affine map, to a circle centered at the origin and an axis-aligned ellipse. In this form, we can detect whether the circle and ellipse are separated, overlapping, or one of the objects is contained in the other. The algorithm resorts to root finding, but the function that provides the roots is of a form that supports robust computing via bisection or Newton's method. Moreover, root bounding is trivial for this function.

The first ellipse in standard form is

$$(\mathbf{X} - \mathbf{K}_0)^{\mathsf{T}} M_0 (\mathbf{X} - \mathbf{K}_0) = 1 \tag{19}$$

The center of the ellipse is $\mathbf{K}_0$. The matrix $M_0$ is positive definite and can be factored as $M_0 = R_0 D_0 R_0^{\mathsf{T}}$, where $R_0$ is a rotation matrix and $D_0$ is a diagonal matrix whose diagonal entries are positive. The diagonal values of $D_0$ are eigenvalues of $M_0$ and the columns of $R_0$ are eigenvectors of $M_0$. Similarly, the second ellipse is defined by

$$(\mathbf{X} - \mathbf{K}_1)^{\mathsf{T}} M_1 (\mathbf{X} - \mathbf{K}_1) = 1 \tag{20}$$

where $\mathbf{K}_1$ is the center and $M_1$ is positive definite. The matrix factors to $M_1 = R_1 D_1 R_1^{\mathsf{T}}$, where $R_1$ is a rotation matrix and $D_1$ is a diagonal matrix whose diagonal entries are positive.

## 4.1 Reduction to Circle and Axis-Aligned Ellipse

The first change of variables is $\mathbf{Y} = D_0^{1/2} R_0^{\mathsf{T}} (\mathbf{X} - \mathbf{K}_0)$, where $D_0^{1/2}$ is the diagonal matrix whose diagonal entries are the square roots of the diagonal entries of $D_0$. Equation (19) transforms to

$$\mathbf{Y}^{\mathsf{T}}\mathbf{Y} = 1 \tag{21}$$

which represents a circle of radius 1 centered at the origin. Defining $\mathbf{K}_2 = D_0^{1/2} R_0^{\mathsf{T}} (\mathbf{K}_1 - \mathbf{K}_0)$ and $M_2 = D_0^{-1/2} R_0^{\mathsf{T}} R_1 D_1 R_1^{\mathsf{T}} R_0 D_0^{-1/2}$, equation (20) transforms to

$$(\mathbf{Y} - \mathbf{K}_2)^{\mathsf{T}} M_2 (\mathbf{Y} - \mathbf{K}_2) = 1 \tag{22}$$

The matrix $M_2$ may be factored using an eigendecomposition to $M_2 = RDR^{\mathsf{T}}$, where $D$ is a diagonal matrix and $R$ is a rotation matrix.

The second change of variables is $\mathbf{Z} = R^{\mathsf{T}}\mathbf{Y}$. Equation (21) is transformed to

$$\mathbf{Z}^{\mathsf{T}}\mathbf{Z} = 1 \tag{23}$$

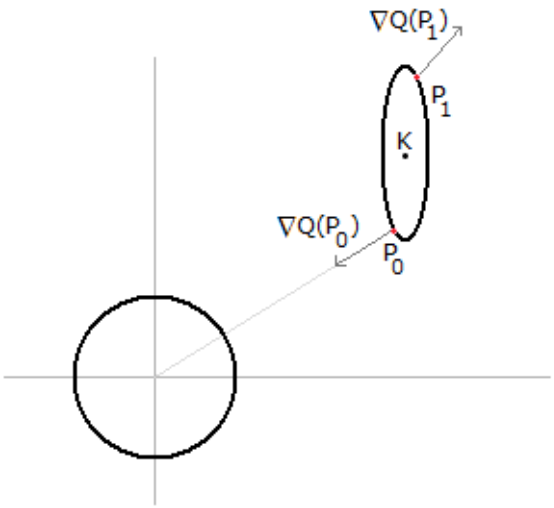
which is the same circle as that of equation (21) but rotated about the origin. Defining $\mathbf{K} = R^{\mathsf{T}}\mathbf{K}_2$, equation (22) is transformed to

$$(\mathbf{Z} - \mathbf{K})^{\mathsf{T}} D(\mathbf{Z} - \mathbf{K}) = 1 \tag{24}$$

Because $D$ is diagonal, this equation represents an axis-aligned ellipse with center point $\mathbf{K}$, as illustrated in Figure 3. The ellipse is implicitly defined by $Q(\mathbf{Z}) = (\mathbf{Z} - \mathbf{K})^{\mathsf{T}} D(\mathbf{Z} - \mathbf{K}) - 1 = 0$, and the gradient vector $\nabla Q(\mathbf{Z})$ is an outer-pointing normal to the point $\mathbf{Z}$ on the ellipse.

---

**Figure 3.** The transformed configuration to a circle of radius 1 with center at the origin and an axis-aligned ellipse. The circle and ellipse are separated. The point $\mathbf{P}_0$ is the ellipse point that is closest to the origin. An outer-pointing normal at this point is $\nabla Q(\mathbf{P}_0)$. The point $\mathbf{P}_1$ is the ellipse point that is farthest from the origin. An outer-pointing normal at this point is $\nabla Q(\mathbf{P}_1)$.




---

The figure shows the case when the ellipse center $\mathbf{K}$ is outside the circle. If $\mathbf{K}$ is inside the circle, then we know that the original ellipses overlap. When $\mathbf{K} = \mathbf{0}$, the circle and ellipse have the same center and necessarily overlap. To determine whether there is full containment, it is sufficient to examine the minimum and maximum of the reciprocals of the diagonal entries of $D$.

## 4.2 Computing the Extreme Points P

To determine the relationship between the ellipse and circle, we need to compute the extreme points $\mathbf{P}$ on the ellipse that are closest and farthest from the origin.

The point $\mathbf{P}_0$ shown in Figure 3 is the ellipse point closest to the origin. We know that $\mathbf{P}_0$ is on the ellipse, so $Q(\mathbf{P}_0) = 0$. An outer-pointing normal to the ellipse at $\mathbf{P}_0$ is the gradient vector $\nabla Q(\mathbf{P}_0) = 2D(\mathbf{P}_0 - \mathbf{K})$. In fact, this vector is parallel to $\mathbf{P}_0$ but in the opposite direction; thus, there is some scalar $s < 0$ for which

$$sD(\mathbf{P}_0 - \mathbf{K}) = \mathbf{P}_0 \tag{25}$$

If $\mathbf{P}_0 = (p_0, p_1)$, $\mathbf{K} = (k_0, k_1)$, and $D = \text{Diag}(d_0, d_1)$ with $d_1 \geq d_0$, then equation (25) may be solved for

$$p_0 = \frac{d_0 k_0 s}{d_0 s - 1}, \quad p_1 = \frac{d_1 k_1 s}{d_1 s - 1} \tag{26}$$

Knowing that $\mathbf{P}_0$ is on the ellipse, we have

$$0 = Q(\mathbf{P}_0) = \frac{d_0 k_0^2}{(d_0 s - 1)^2} + \frac{d_1 k_1^2}{(d_1 s - 1)^2} - 1 \tag{27}$$

It turns out that equation (27) has a unique negative root $s_0$. Similarly, there is a scalar $s > 0$ such that $sD(\mathbf{P}_1 - \mathbf{K}) = \mathbf{P}_1$ and $s$ is also a solution to equation (27). The equation can have multiple positive roots, but the one corresponding to $\mathbf{P}_1$ is the largest positive root.

Generally, $\mathbf{P}_0$ can be inside or outside the circle, so we do not know ahead of time that the smallest root $s$ is negative. Equation (27) may be rewritten as a quartic polynomial equation: $p(s) = (d_0 s - 1)^2 (d_1 s - 1)^2 - d_0 k_0^2 (d_1 s - 1)^2 - d_1 k_1^2 (d_0 s - 1)^2 = 0$. The closed-form equations for the roots of a quartic may be used to compute the real-valued roots. If you have a robust polynomial root finder, you may use it to compute the candidate $s$-values that generate $\mathbf{P}_0$ and $\mathbf{P}_1$.

An algorithm that is better suited for numerical computations is described next. Define the function

$$f(s) = \frac{d_0 k_0^2}{(d_0 s - 1)^2} + \frac{d_1 k_1^2}{(d_1 s - 1)^2} - 1 \tag{28}$$

The sets of roots of $p(s)$ and $f(s)$ are the same. The first derivative is

$$f'(s) = \frac{-2 d_0^2 k_0^2}{(d_0 s - 1)^3} + \frac{-2 d_1^2 k_1^2}{(d_1 s - 1)^3} \tag{29}$$

and the second derivative is

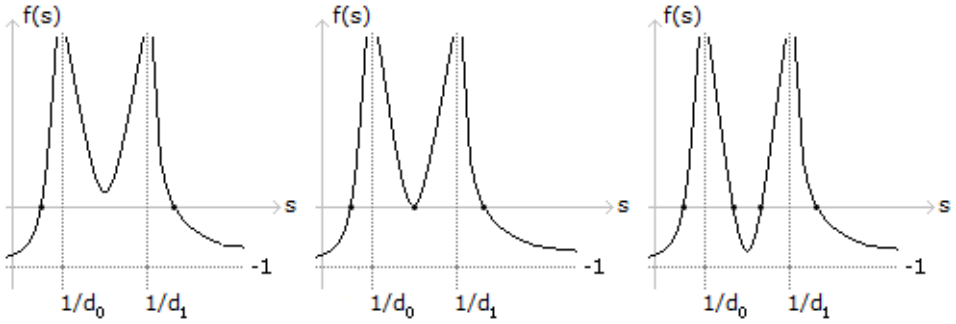
$$f''(s) = \frac{6 d_0^3 k_0^2}{(d_0 s - 1)^4} + \frac{6 d_1^3 k_1^2}{(d_1 s - 1)^4} > 0 \tag{30}$$

The second derivative is always positive, which makes $f(s)$ a *convex function* on each interval for which $f$ is defined. Such functions are ideal for locating roots using Newton's method. Alternatively, the function is of a form that is convenient for using the more conservative bisection method for root finding.

In the special case $d_0 = d_1$, the function reduces to $f(s) = 2 d_0 k_0^2 / (d_0 s - 1)^2 - 1$, a form which makes it easy to extract the roots.

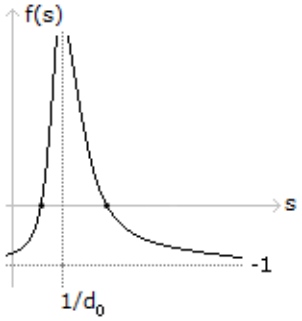
Typical graphs of $f(s)$ for $d_0 > d_1$ are shown in Figure 4.

**Figure 4.** Typical graphs of $f(s)$ when $d_0 > d_1$. The smallest root corresponds to $\mathbf{P}_0$. The other roots are always positive and the largest of these corresponds to $\mathbf{P}_1$.




A typical graph of $f(s)$ for $d_0 = d_1$ is shown in Figure 5.

**Figure 5.** The typical graph of $f(s)$ when $d_0 = d_1$, in which case $f(s) = 2d_0k_0^2/(d_0s - 1)^2 - 1$. The smallest root corresponds to $\mathbf{P}_0$ and the largest root is always positive and corresponds to $\mathbf{P}_1$.




It is clear from the graphs what the root-bounding intervals are when trying to select an initial guess for Newton's method or when trying to bisect to find a root. When the ellipse is separated from the circle, $\mathbf{P}_0$ is outside the circle, in which case $f(0) = d_0k_0^2 + d_1k_1^2 - 1 = Q(0) > 0$. From the geometry, this forces the smallest root of $f(s)$ to be negative and correspond to $\mathbf{P}_0$. It is also easy to demonstrate that $\lim_{|s|\to\infty} f(s) = -1$, which the figures illlustrate.

Generally, we can compute the roots of $f(s)$ and for each corresponding point $\mathbf{P}$, compute the distance from the origin, keeping track of the minimum distance and the maximum distance. The relationship of the circle (generated from ellipse 0) and the axis-aligned ellipse (generated from ellipse 1) is then

```
if (maxDistance < 1)
{
    return ELLIPSE0_STRICTLY_CONTAINS_ELLIPSE1;
}
else if (maxDistance > 1)
{
```

```
        if (minDistance < 1)
        {
            // The ellipses overlap but each ellipse contains points the other
            // ellipse does not contain.
            return ELLIPSES_OVERLAP;
        }
        else if (minDistance > 1)
        {
            if (circle center outside the ellipse)  // f(0) > 0
            {
                return ELLIPSES_SEPARATED;
            }
            else
            {
                return ELLIPSE1_STRICTLY_CONTAINS_ELLIPSE0;
            }
        }
        else  // minDistance = 1
        {
            // The solid ellipses have a single point of intersection.
            if (circle center outside the ellipse)  // f(0) > 0
            {
                return ELLIPSE0_OUTSIDE_ELLIPSE1_BUT_TANGENT;
            }
            else
            {
                return ELLIPSE1_CONTAINS_ELLIPSE0_BUT_TANGENT;
            }
        }
    }
    else  // maxDistance = 1
    {
        if (minDistance < 1)
        {
            // The solid ellipses have a single point of intersection.
            return ELLIPSE0_CONTAINS_ELLIPSE1_BUT_TANGENT;
        }
        else // minDistance = 1
        {
            return ELLIPSES_EQUAL;
        }
    }
```

### 4.3 Computing the Roots of $f(s)$

In the case when $d_0 = d_1$, we have $f(s) = 2d_0k_0^2/(d_0s-1)^2-1$. The function is zero when $s = (1\pm\sqrt{2d_0k_0^2})/d_0$. A similar root finding occurs when $d_0 > d_1$ but $k_0 = 0$ or $k_1 = 0$.

In the case when $d_0 > d_1$, $k_0 \neq 0$, and $k_1 \neq 0$, we have $f(s) = d_0k_0^2/(d_0s-1)^2 + d_1k_1^2/(d_1s-1)^2 - 1$. We need root bounds for the various configurations shown in Figure 4.

Let us consider first the smallest root $\bar{s}$ of $f(s)$, a number in the interval $(-\infty, 1/d_0)$. We could bisect the root using floating-point arithmetic where the interval minimum is chosen to be the smallest finite floating-point number. However, we can construct a better estimate than that. If $f(0) \leq 0$, then the root-bounding interval is $[0, 1/d_0]$. If $0 < f(0) = d_0k_0^2 + d_1k_1^2 - 1$, then $\bar{s} < 0$. We know that $d_0 > d_1$, so for $s < 0$, $d_0s - 1 < d_1s - 1 < 0$ and $1/(d_0s-1)^2 < 1/(d_1s-1)^2$ and

$$f(s) = \frac{d_0k_0^2}{(d_0s-1)^2} + \frac{d_1k_1^2}{(d_1s-1)^2} - 1 < \frac{d_0k_0^2 + d_1k_1^2}{(d_1s-1)^2} - 1 \tag{31}$$

The right-hand side is negative for $s < (1 - \sqrt{1 + f(0)})/d_1 < 0$. The interval $[(1 - \sqrt{1 + f(0)})/d_1, 0]$ bounds the root $\bar{s}$ and may be used for bisection.

A similar argument applies for the largest root $\hat{s}$ of $f(s)$, a number in the interval $(1/d_1, +\infty)$. We know that $d_0 > d_1$, so for $s > 1/d_1$, $d_0 s - 1 > d_1 s - 1 > 0$ and $1/(d_0 s - 1)^2 < 1/(d_1 s - 1)^2$ and equation (31) is valid. The right-hand side is negative for $s > (1 + \sqrt{1 + f(0)})/d_1$. The interval $(1/d_1, (1 + \sqrt{f(0)})/d_1]$ bounds the root $\hat{s}$ and may be used for bisection.

The function $f(s)$ has 0, 1, or 2 roots in the interval $(1/d_0, 1/d_1)$, depending on the sign of $f(\tilde{s})$ where $f'(\tilde{s}) = 0$. We can compute $\tilde{s}$ directly by solving $f'(s) = 0$ for $\tilde{s} = (1 + \rho)/(d_0 + d_1\rho)$ where $\rho = (d_0 k_0/(d_1 k_1))^{2/3}$. If $f(\tilde{s}) > 0$, there are no roots in $(1/d_0, 1/d_1)$. If $f(\tilde{s}) = 0$, $\tilde{s}$ is the only root in the interval. If $f(\tilde{s}) < 0$, there is a unique root in the interval $(1/d_0, \tilde{s})$ and a unique root in the interval $(\tilde{s}, 1/d_1)$. Each of these roots may be computed by bisecting the corresponding interval.

Once all roots are computed for $f(s)$, we compute $\mathbf{P} = (d_0 k_0 s/(d_0 s - 1), d_1 k_1 s/(d_1 s - 1))$ for each root $s$ and then compute the squared distance $|\mathbf{P}|^2$. The minimum and maximum squared distances are computed and the classification mentioned previously determines the relationship of the ellipses.

## 4.4 An Implementation

In GTEngine, the test-intersection query is implemented in the file GteIntrEllipse2Ellipse2.h.